The NetLogger Toolkit

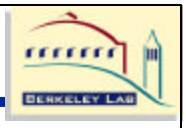
Brian L. Tierney
Dan Gunter

Data Intensive Distributed Computing Group
Lawrence Berkeley National Laboratory

Overview

- The Problem
  - When building distributed systems, we often observe unexpectedly low performance
    - the reasons for which are usually not obvious
  - The bottlenecks can be in any of the following components:
    - the applications
    - the operating systems
    - the disks or network adapters on either the sending or receiving host
    - the network switches and routers, and so on
- The Solution:
    - Highly instrumented systems with precision timing information and analysis tools

Bottleneck Analysis



- Distributed system users and developers often assume the problem is network congestion
  - This is often not true
- In our experience tuning distributed applications, performance problems are due to:
  - network problems: ~40%
  - host problems: ~20%
  - application design problems/bugs: ~40%
    - 50% client , 50% server
- Therefore it is equally important to instrument the applications

NetLogger Toolkit

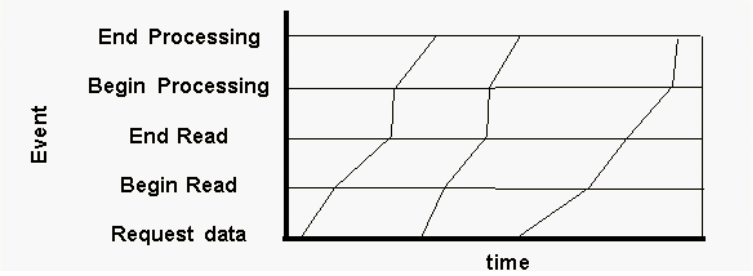
- We have developed the NetLogger Toolkit (short for Networked Application Logger), which includes:
  - tools to make it easy for distributed applications to log interesting events at every critical point
  - tools for host and network monitoring
- The approach is novel in that it combines network, host, and application-level monitoring to provide a complete view of the entire system.
- This has proven invaluable for:
  - isolating and correcting performance bottlenecks
  - debugging distributed applications

NetLogger Components

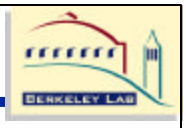
- NetLogger Toolkit contains the following components:
  - NetLogger message format
  - NetLogger client library (C, C++, Java, Perl, Python)
  - NetLogger visualization tools
  - NetLogger host/network monitoring tools
- Source code and binaries are available at:
  - http://www-didc.lbl.gov/NetLogger/
- Additional critical component for distributed applications:
  - NTP (Network Time Protocol) or GPS host clock is required to synchronize the clocks of all systems

Key Concepts

- NetLogger visualization tools are based on time correlated and/or object correlated events.
- NetLogger client libraries include:
  - precision timestamps (default = microsecond)
  - ability for applications to specify an “object ID” for related events, which allows the NetLogger visualization tools to generate an object “lifeline”

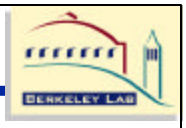


NetLogger Message Format



- We are using the IETF-developed Universal Logger Message (ULM) format:
  - a list of “field=value” pairs
  - required fields: DATE, HOST, PROG, and LVL
    - —DATE = YYYYMMDDHHSS.SSSSSS
    - —PROG: program name
    - —LVL is the severity level (Emergency, Alert, Error, Usage, etc.)
  - followed by optional user defined fields
  - see: http://www-didc.lbl.gov/NetLogger/draft-abela-ulm-05.txt
- NetLogger adds this required fields:
  - NL.EVNT, a unique identifier for the event being logged
    - —e.g.: SERVER_IN, VMSTAT_USER_TIME, NETSTAT_RETRANSSEG

NetLogger Message Format

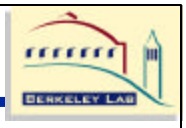


- Sample NetLogger ULM event:

```
DATE=19980430133038.055784 HOST=foo.lbl.gov
  PROG=testprog LVL=Usage NL.EVNT=SEND_DATA
  SEND.SZ=49332
```

  - This says program named *testprog* on host *foo.lbl.gov* performed event named SEND_DATA, size = 49332 bytes, at the time given
- User-defined data elements (any number) are used to store information about the logged event - for example:
  - NL.EVNT=SEND_DATA SEND.SZ=49332
    - —the number of bytes of data sent
  - NL.EVNT=NETSTAT_RETRANSSEGS NS.RTS=2
    - —the number of TCP retransmits since the previous event

When to use NetLogger



- When you want to:
  - do performance/bottleneck analysis on distributed applications
  - determine which hardware components to upgrade to alleviate bottlenecks
  - do real-time or post-mortem analysis of applications
  - correlate application performance with system information (ie: TCP retransmission's)
- works best with applications where you can follow a specific item (data block, message, object) through the system

When NOT to use NetLogger

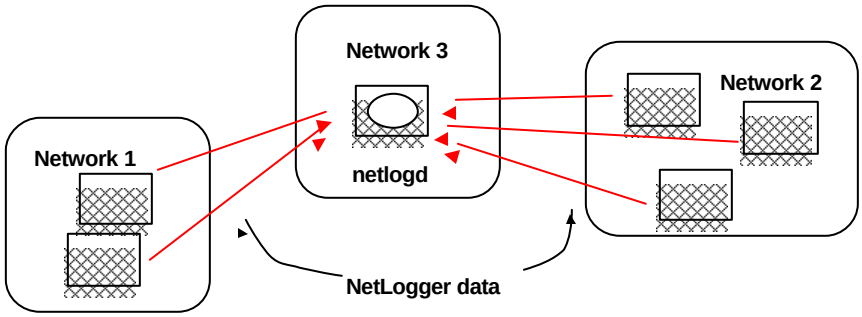
- Analyzing massively parallel programs (e.g.: MPI)
  - Current visualization tools don't scale beyond tracking about 20 types of events at a time
- Analyzing many very short events
  - system will become overwhelmed if too many events
  - we typically use NetLogger to monitor events that take > .5 ms
  - e.g: probably don't want to use to instrument the UNIX kernel

NetLogger API

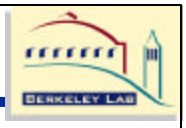
- NetLogger Toolkit includes application libraries for generating NetLogger messages
  - Can send log messages to:
    - file
    - host/port (*netlogd*)
    - syslogd
    - memory, then one of the above
- C, C++, Java, Fortran, Perl, and Python APIs are currently supported

netlogd

- Use *netlogd* to collect NetLogger messages at a central host
  - use to avoid the need to sort/merge several log files from several places

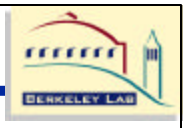


Logging to Memory



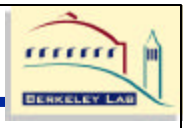
- The NetLogger client library includes an option to buffer log messages in memory:
  - useful if monitoring bursts of events with a duration < 1 ms
- Flushing of events to disk or network will occur:
  - automatically when specified memory block full
  - when calling NetLoggerFlush()
  - when calling NetLoggerClose()
- Size of memory buffer specified by NL_MAX_BUFFER in netlogger.h
  - default = 10,000 messages (typical message size is 128 bytes)

NetLogger API



- Only 6 simple calls:
  - NetLoggerOpen()
    - create NetLogger handle
  - NetLoggerWrite()
    - get timestamp, build NetLogger message, send to destination
  - NetLoggerGTWrite()
    - must pass in results of Unix gettimeofday() call
  - NetLoggerFlush()
    - flush any buffered message to destination
  - NetLoggerSetLevel()
    - set ULM severity level
  - NetLoggerClose()
    - destroy NetLogger handle

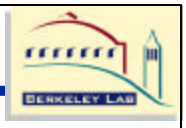
NetLogger Open Call



NLhandle *lp = NULL;
lp = NetLoggerOpen(char *program_name, char *dest_url, int flags);

- program_name: name to be inserted into ULM “program” field
- dest_url: destination of log file; valid URLs formats include:
  - file://path/file
  - x-netlog://host:port
  - x-syslog://localhost
- flags: bitwise “or” of the following:
  - NL_MEM: buffer in memory
  - NL_ENV: destination must be specified by the NL_DEST_ENV environment variable; NetLogger is off if this variable not found

NetLoggerOpen() shell environment variables



- Enable/Disable logging:
  `setenv NETLOGGER_ON {true, on, yes, 1}:` do logging
  `setenv NETLOGGER_ON {false, off, no, 0}:` do not do logging
- Log Destination: **`setenv NL_DEST_ENV logging destination`**

Examples:

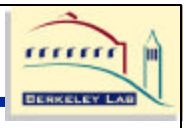
`setenv NL_DEST_ENV file://tmp/netlog.log`
write log messages to file /tmp/netlog.log

`setenv NL_DEST_ENV x-netlog://loghost.lbl.gov`
send log messages to netlogd on host loghost.lbl.gov, default port

`setenv NL_DEST_ENV x-netlog://loghost.lbl.gov:6006`
send log messages to netlogd on host loghost.lbl.gov, port 6006

- NL_DEST_ENV overrides the URL passed in via the NetLoggerOpen() call.

Typical Use



- Using the environment variables, application and middleware developers don't have to worry about command line arguments or middleware APIs to enable/disable logging.
- Example: middleware includes the following call:

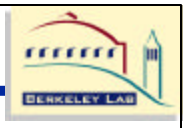
```
NetLoggerOpen("globus", NULL, NL_ENV);
```

  - Default behavior: logging is off
  - If user sets "NL_DEST_ENV" to a valid log destination, then logging will be turned on
- Example: client includes the following call:

```
NetLoggerOpen("my_app", "file://tmp/myapp.log", 0);
```

  - Default behavior: logging is on
  - If user sets: **`NETLOGGER_ON = off`**: Logging is disabled

NetLogger Write Call



- Creates and Writes the log event:

```
NetLoggerWrite(nl, "EVENT_NAME",
  "EVENTID=%d F2=%d F3=%s F4=%.2f", id,
  user_data, user_string, user_float);
```

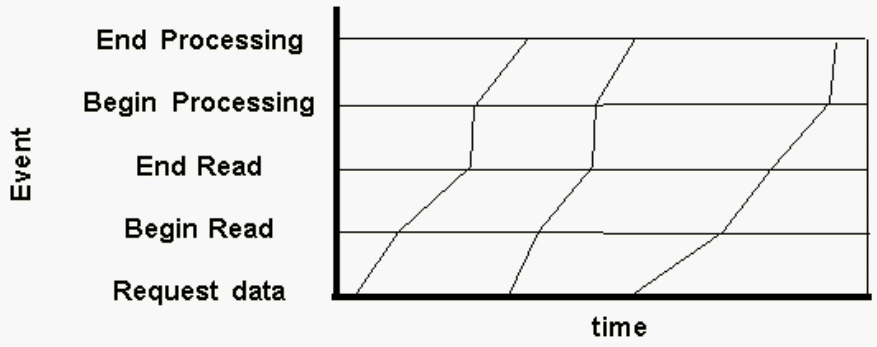
  - timestamps are automatically done by library
  - the "event name" field is required, all other fields are optional
  - this call is thread-safe: automatically does a mutex lock around write call (compile time option)
- Example:

```
NetLoggerWrite(nl, "HTTPD.START_DISK_READ",
  "HTTPD.FNAME=%s HTTPD.HOST=%s", fname,
  hostname);
```

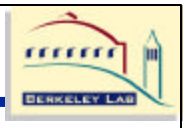
Sample NetLogger Use

```
lp = NetLoggerOpen( progname,
          x-netlog://loghost.lbl.gov, 0);

while (!done)
{
       NetLoggerWrite(lp, "EVENT_START",
                      "TEST.SIZE=%d", size);

       /* perform the task to be monitored */
       done = do_something(data, size);

       NetLoggerWrite(lp, "EVENT_END");
}
NetLoggerClose(lp);
```


NetLogger Event "Life Lines"

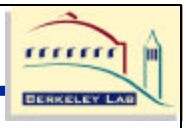


Event ID



- In order to associate a group of events into a “lifeline”, you must assign an event ID to each NetLogger event
- Sample Event Ids
  - file name
  - block ID
  - frame ID
  - user name
  - host name
  - *combination of the above*
  - etc.

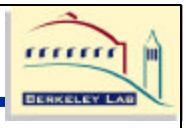
Sample NetLogger Use with Event IDs



```
lp = NetLoggerOpen(progname, NULL, NL_ENV);
for (i=0; i< num_blocks; i++)  {
    NetLoggerWrite(lp, “START_READ”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    read_block(i);
    NetLoggerWrite(lp, “END_READ”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    NetLoggerWrite(lp, “START_PROCESS”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    process_block(i);
    NetLoggerWrite(lp, “END_PROCESS”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    NetLoggerWrite(lp, “START_SEND”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    send_block(i);
    NetLoggerWrite(lp, “END_SEND”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
}
NetLoggerClose(lp);
```


1.

NetLogger Host/Network Tools



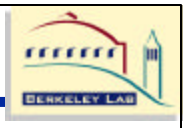
- Wrapped UNIX network and OS monitoring tools to log "interesting" events using the same log format
  - *netstat* (TCP retransmissions, etc.)
  - *vmstat* (system load, available memory, etc.)
  - *iostat* (disk activity)
  - *ping*
- These tools have been wrapped with Perl programs which:
  - parse the output of the system utility
  - build NetLogger messages containing the results

Sample NetLogger System Monitoring Tool

- Example: nl_vmstat -t 60 -d 5000 -m 2 logger.lbl.gov
  - Perl program will exec *vmstat* every 5 seconds for 1 hour, and send the results to *netlogd* on host logger.lbl.gov
  - Generates the following information:
    - CPU usage by User
    - CPU usage by System
- NetLogger Messages:

```
DATE=19990706125055.891620 HOST=portnoy.lbl.gov
  PROG=nl_vmstat LVL=Usage NL.EVNT=VMSTAT_USER_TIME
  VMS.VAL=9
DATE=19990706125055. 891112 HOST=portnoy.lbl.gov
  PROG=nl_vmstat LVL=Usage NL.EVNT=VMSTAT_SYS_TIME
  VMS.VAL=5
```

NetLoggerized tcpdump



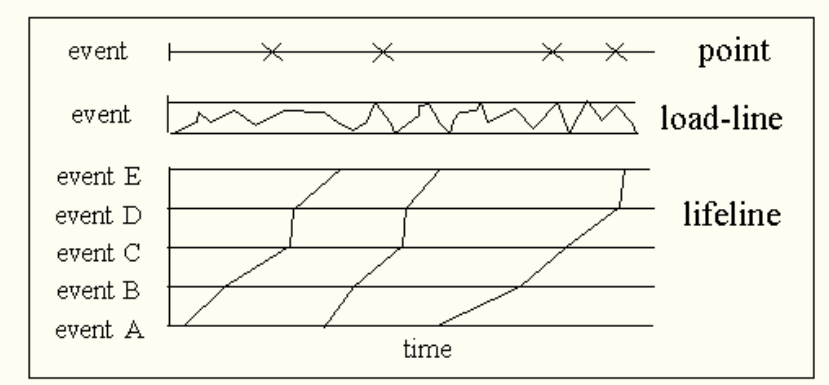
- Precise real-time monitoring of TCP events on a per stream bases
  - TCP retransmits
  - TCP window size
- Example:
  - **tcpdump -A tcp and host piggy.ittc.ukans.edu and port 23**
- Generates the following NetLogger data:
  - **DATE=20000419171039.78654 HOST=piggy.ittc.ukans.edu PROG=tcpdump LVL=ErrorNL.EVNT=TCPD_REXSEG SN=145 SRC_HOST=falcon.cc.ukans.edu SRC_PORT=23 DST_HOST=piggy.ittc.ukans.edu DST_PORT=2800**
- http://www.ittc.ukans.edu/projects/enable/tcpdump

NetLogger Visualization Tools

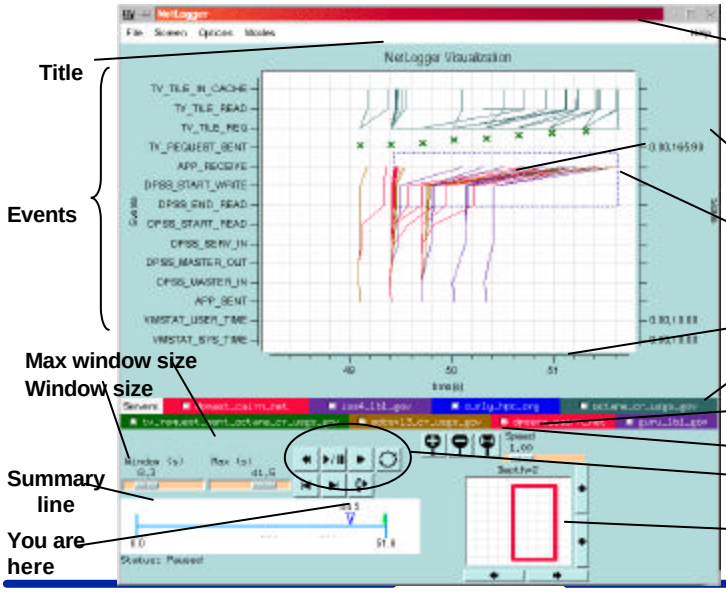
- Exploratory, interactive analysis of the log data has proven to be the most important means of identifying problems
  - this is provided by *nlv* (NetLogger Visualization)
- *nlv* functionality:
  - can display several types of NetLogger events at once
  - user configurable: which events to plot, and the type of plot to draw (lifeline, load-line, or point)
  - play, pause, rewind, slow motion, zoom in/out, and so on
  - *nlv* can be run post-mortem or in real-time
    - real-time mode done by reading the output of *netlogd* as it is being written

NLV Graph Types

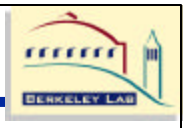
- **nlv supports graphing of "points", load-lines, and lifelines**



NLV Screenshot



NLV Configuration

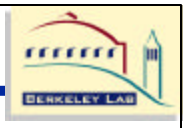


- NLV is very flexible, with many options settable in the configuration file.
- Format:

```
eventset +/-eventset_name {
 { type <line,point,load> }
 { id { list of ULM field names used to determine which
  NetLogger messages get grouped into the same graph
  primitive } }
 { group { list of ULM field names which will be mapped to
  the same color } }
 { val field_name min_val max_val }
 { annotate { list of field names to display in with annotate
  option } }
 { events { list of all event ID's in this lifeline } }
}
```

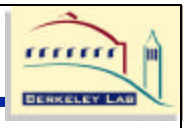
- Each nlv graph object needs to be defined by an "eventset"
- Events and event-sets both use "+" and "-" to indicate default (i.e. on startup) visibility

Example NLV Configuration



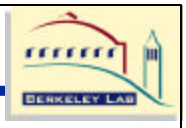
```
# display vmstat info as a "loadline"
eventset +VMSTAT {
{ type load }
# loadline constructed from messages with the same HOST and NL.EVNT
{ id { HOST NL.EVNT } }
# messages with the same HOST get the same color
{ group HOST }
#list of NL.EVNT values in this set_
{ events { +VMSTAT_SYS_TIME +VMSTAT_USER_TIME } }
}

# display netstat TCP retransmits as a "point"
eventset +NETSTAT {
{ type point }
# ignore values outside the range 0 to 999
{ val NS.VAL 0.0 999.0 }
# point constructed from messages from the same HOST and PROG
{ id { HOST PROG } }
# messages with the same HOST get the same color
{ group HOST }
{ events { +NETSTAT_RETRANSSEGS } }
}
```

Example NLV Configuration



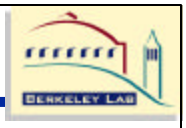
```
# display server data as a "lifeline"
eventset +SERVER_READ {
{ type line }

# lifeline constructed from messages from the same client
  and server
{ id { CLIENT_HOST DPSS.SERV } }

# messages with the same DPSS.SERV get the same color
{ group DPSS.SERV }

{ events {
  +APP_SENT
  +DPSS_SERV_IN
  +DPSS_START_READ
  +DPSS_END_READ
  +DPSS_START_WRITE
  +APP_RECEIVE }
}
}
```


How to Instrument Your Application

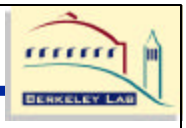


- You'll probably want to add a NetLogger event to the following places in your distributed application:
  - before and after all disk I/O
  - before and after all network I/O
  - entering and leaving each distributed component
  - before and after any significant computation
    - e.g.: an FFT operation
  - before and after any significant graphics call
    - e.g.: certain CPU intensive OpenGL calls
- This is usually an iterative process
  - add more NetLogger events as you zero in on the bottleneck

Does NetLogger affect application performance?



- Only if you use it incorrectly or log too much
- There are several things to be careful of when doing this type of monitoring:
  - If logging to disk, don't log to a nfs mounted disk
    - best to log to /tmp, which may actually be RAM (Solaris)
  - Probably don't want to send log messages to a slow (i.e.: 10BT) or congested network, as you'll just make it worse
    - log to a local file instead
- Sample NetLoggerWrite Performance: 100000 calls/sec
  - can make 1000 NetLoggerWrite calls / sec and only effect your application by 1%

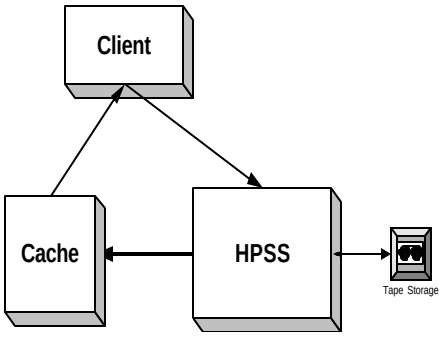


NetLogger Case Studies

Example: HPSS Storage Manager Application

- NetLogger was used to test and verify the results of a Storage Access Coordination System (STACS) by LBNL's Data Management Group
- STACS is designed to optimize the use of a disk cache with an HPSS Mass Storage system, and tries to minimize tape mount requests by clustering related data on the same tape
- NetLogger was used to look at:
  - per-query latencies
  - to show that subsequent fetches of spatially clustered data "hit" in the cache.
- (http://gizmo.lbl.gov/sm/)

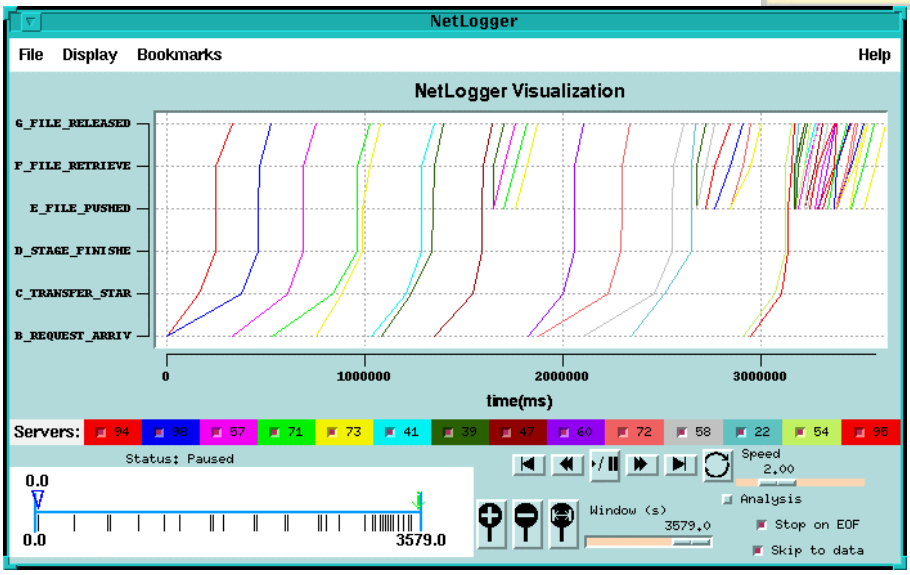
STACS Instrumentation Points



Monitoring Points:

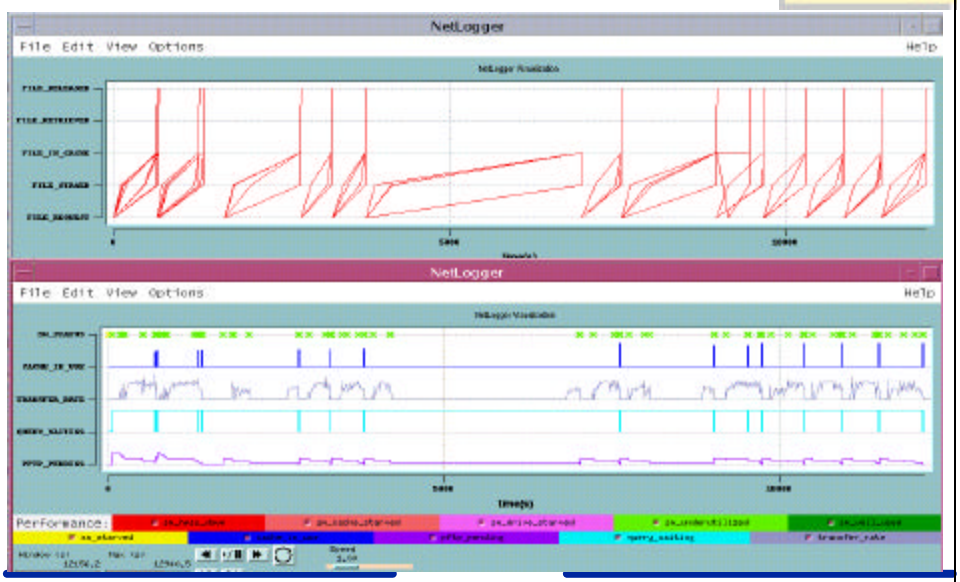
A) request arrives at HPSS
B) start transfer from tape
C) tape transfer finished
D) file available to client
E) file retrieved by client
F) file released by client

NLV for STACS: Tracking File Requests



NetLogger

Tracking Files and System Performance

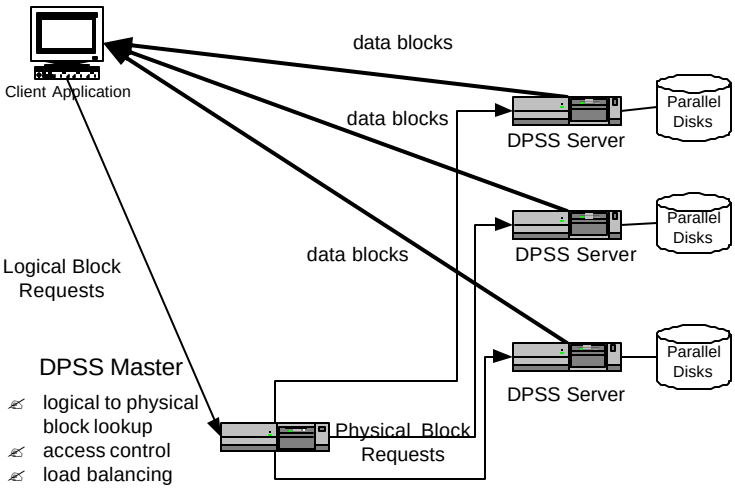


NetLogger

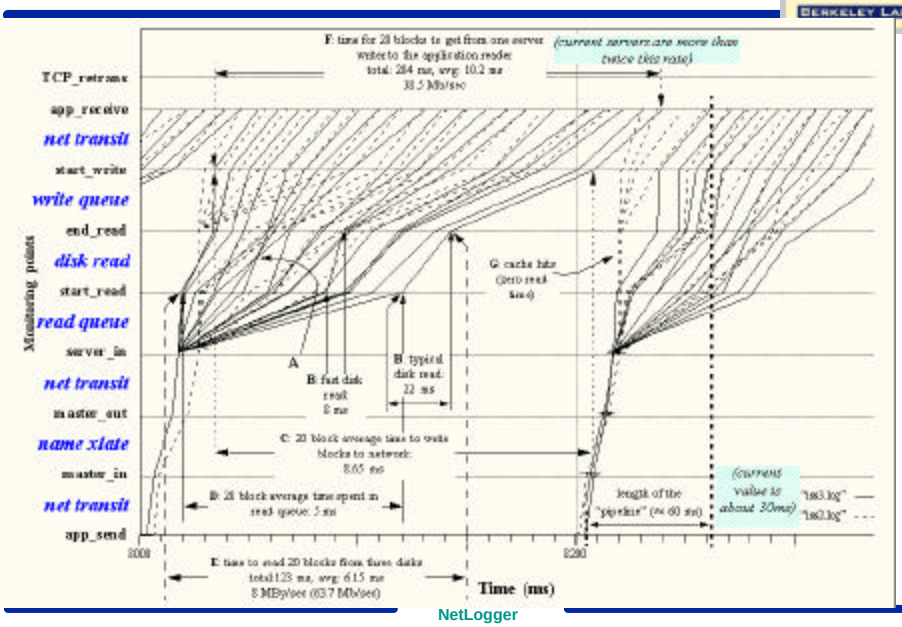
Example: Parallel Data Block Server

- The Distributed Parallel Storage Server (DPSS)
  - provides high-speed parallel access to remote data
  - Unique features of the DPSS:
    - On a high-speed network, can actually access remote data faster that from a local disk
      - —70 MB/sec (DPSS) vs 22 MB/sec (local disk)
    - Only need to send parts of the file currently required over the network
      - —e.g.: client may only need 100 MB from a 2 GB data set
      - —analogous to http model
- NetLogger was used for performance tuning and debugging of the DPSS

DPSS Cache Architecture

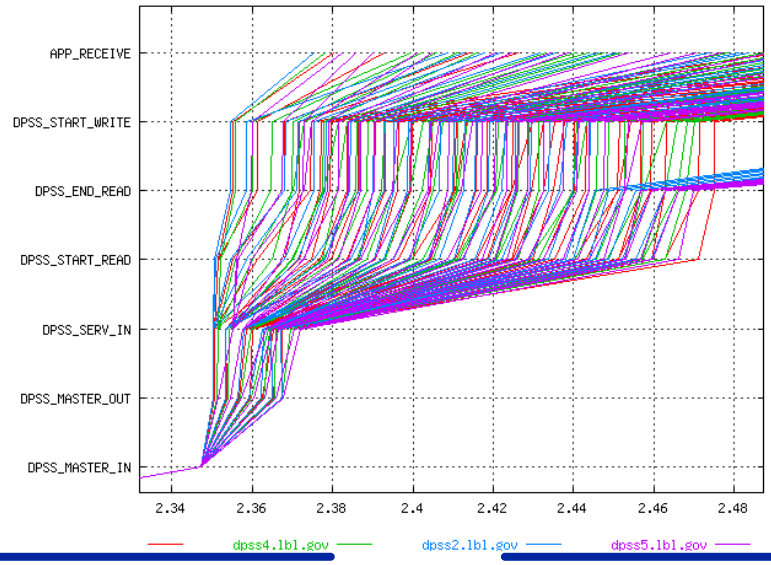


NetLogger Results for the DPSS



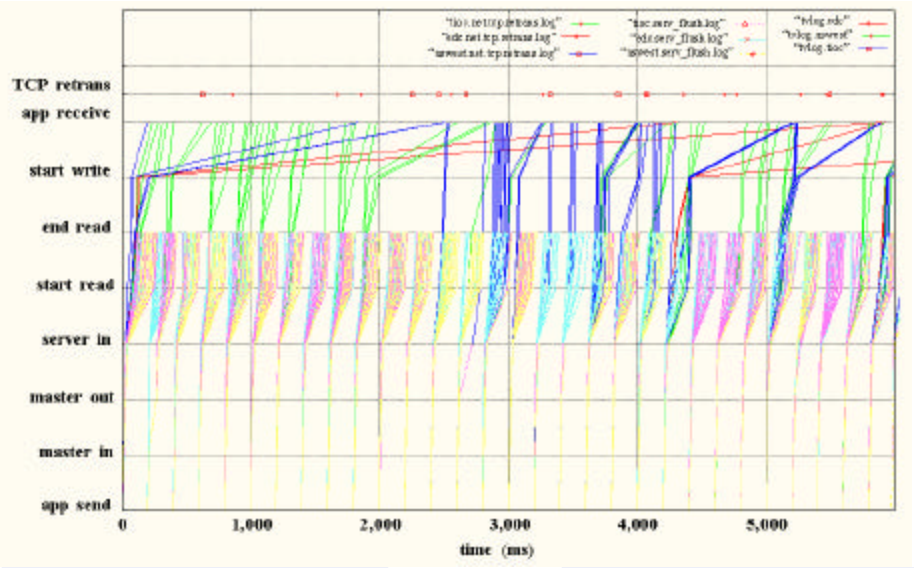
NetLogger

NetLogger Results for the DPSS



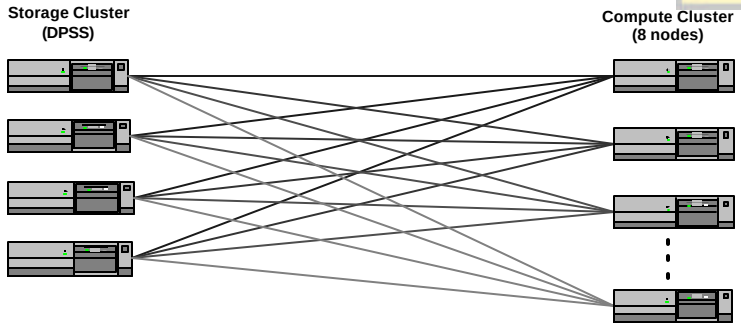
NetLogger

NetLogger Results for the DPSS over a WAN

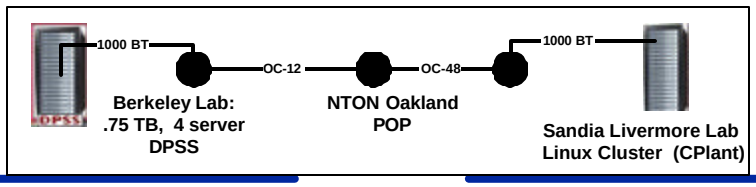


NetLogger

DPSS Performance: Used NetLogger for performance tuning

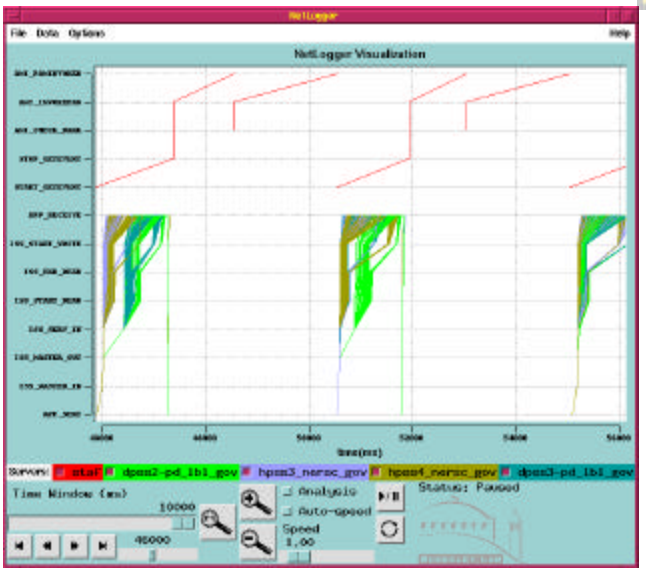


Total Throughput (single dataset to a single cluster application):
570 Mbits/sec (71 MB/sec) on 32 data streams (17 Mbits/sec/stream)



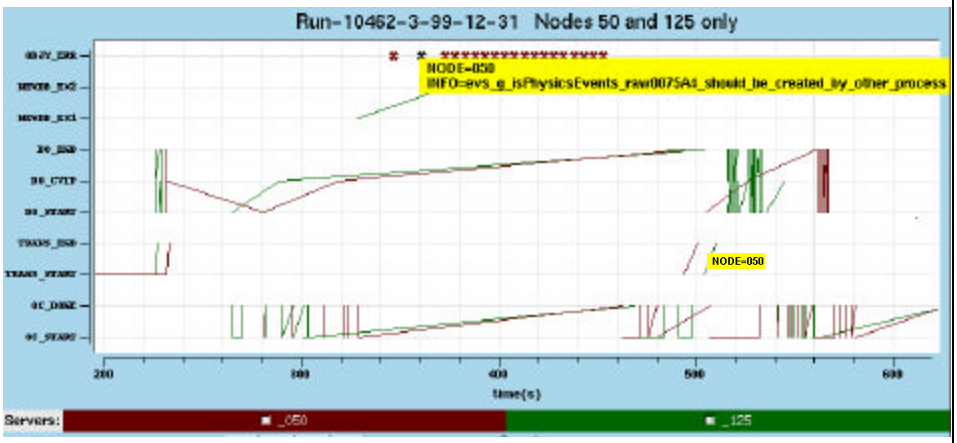
NetLogger

Example: NLV of DPSS with a HENP client



NetLogger

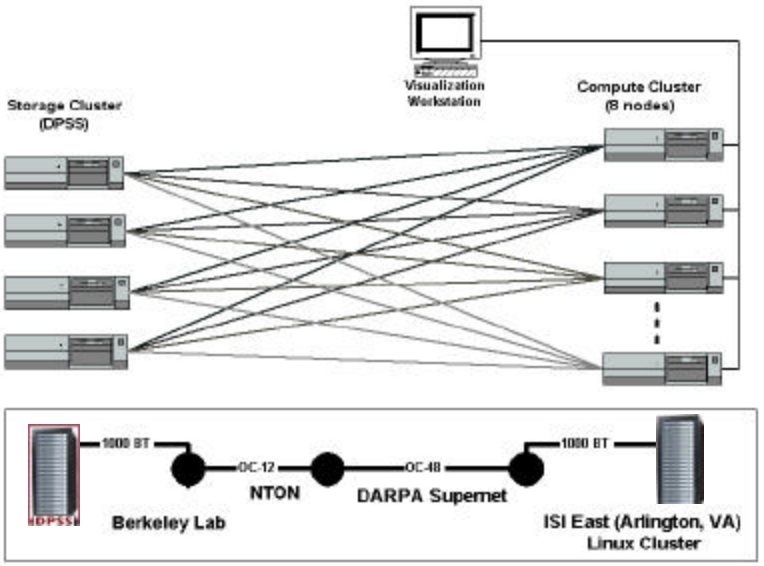
Example: Babar data analysis: 2 nodes with Objectivity Error



NetLogger

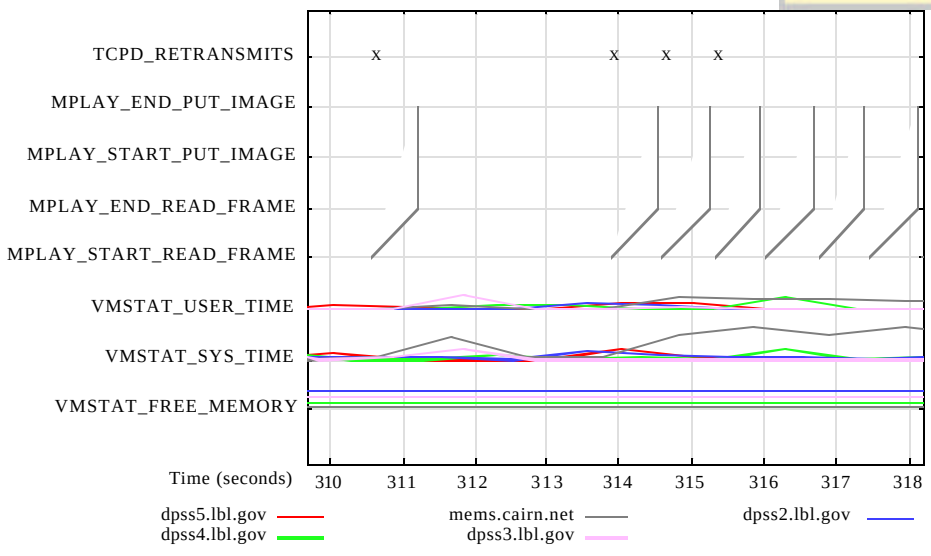
23

Example: Matisse Project



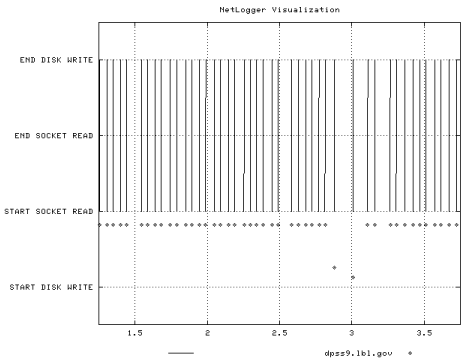
NetLogger

Example: Combined Host and Application Monitoring

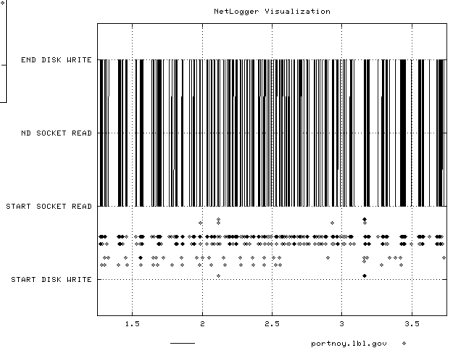


NetLogger

Example: NetLogger of ncftp client



- ncftp client on a 10BT ethernet

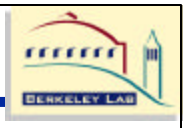


- ncftp client on a 1000BT ethernet

Current/Future NetLogger Work

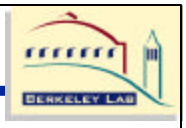
- Binary format (faster!)
- XML format (slower!!)
- Publish/Subscribe API
  - Producer X
    - NetLoggerPublish(“MONITORING_EVENT_NAME”, ...)
  - Consumer Y
    - NetLoggerSubscribe(X, “MONITORING_EVENT_NAME”, ..)

Getting NetLogger



- Source code and binaries are available at:
  - http://www-didc.lbl.gov/NetLogger
- Client libraries run on all Unix platforms
- Solaris, Linux, and Irix versions of *nlv* are currently supported

For More Information



Email:bltierney@lbl.gov

http://www-didc.lbl.gov/NetLogger/
- download NetLogger components
- tutorial
- user guide

http://www-didc.lbl.gov/tcp-wan.html
- links to all network tools mentioned here
- sample TCP buffer tuning code, etc.,